Examining ASP.NET's Membership, Roles, and Profile - Part 10

By Scott Mitchell

Introduction

The Membership system automatically tracks the last date and time each user's account has been accessed. With the `SqlMembershipProvider`, this information is stored in the `aspnet_Users` database table in a `datetime` column named `LastActivityDate`. This column is automatically updated to the current UTC date and time whenever a user logs into the site, whenever their user account information is updated, and whenever their user account information is retrieved.

In addition to tracking each user's last activity date and time, the Membership system includes a method named `GetNumberOfUsersOnline`. This method returns the number of users whose last activity date and time is within a specified window; by default, this method returns the number of users whose `aspnet_Users.LastActivityDate` value falls within the last 15 minutes.

This article examines the `GetNumberOfUsersOnline` method and see how to extend the Membership system to include additional user activity information. Specifically, we will add a new table to the database used by the `SqlMembershipProvider` that associates a description of each user's current action. We will then update our ASP.NET pages to update the records in this table to include a description of the user's current action. For example, when visiting the home page we may use the description, "Viewing the home page." Finally, we will create a web page that displays the list of currently logged on users and their last known action. Read on to learn more!

Tracking the Date and Time of a User's Last Activity

Imagine that you work on a website that supports user accounts and that your boss wants to show on each page how many currently logged in users are visiting the site. In attempting to tackle this problem your first approach might be to create a new database table named `UsersOnline` that contains a single row and column that indicates the number of users currently logged in. For example, when first deployed this table would have one record with a value of 0. Whenever a user logs onto the site through the login page, you would run an `UPDATE` query to increment this lone record's value by 1. Correspondingly, whenever a user clicks the Logoff link, you would run an `UPDATE` query to decrement this record's value by 1. To display the total number of logged in users, then, you would just return and display this table's single numeric value.

The problem with this approach is that users may log off of the site implicitly. That is, rather than logging off by clicking the Logoff button, they may just close their browser window. Consequently, as more and more users log on, but then don't log off by clicking the Logoff link, the `UsersOnline` table will continue to be less and less accurate, reporting hundreds or thousands of logged on users when there may only be a small handful!

In short, it's impossible for your code on the web server to know exactly how many people who have signed into the site are still actively viewing it. A compromise that is commonly used (such as with Session variables) is to define a timeout and to assume anyone who has not accessed the site within the timeout period has logged out.

The Membership system automatically tracks users' last activity dates. With the `SqlMembershipProvider`, this information is stored in the `aspnet_Users` table's `LastActivityDate` column. This column's value is updated to the current UTC date and time value from a variety of actions:

- **Retrieving the user's password.** This action is performed whenever the `Membership.ValidateUser` method is called, which is called by the Login Web control.

- **Updating the user's information.** This action transpires when the `Membership.UpdateUser` method is called. This method is commonly used in user administration pages, such as the administration pages created by Dan Clem in the Rolling Your Own Website Administration Tool article.
- **Retrieving information about the current user.** The `Membership.GetUser` method returns information about a particular user. It also accepts an option Boolean parameter that indicates whether or not to update the selected user's `aspnet_Users.LastActivityDate` value. (By default, this column is updated.)

In addition to tracking users' last activity dates, the Membership system includes a `GetNumberOfUsersOnline` method that returns the number of users whose `aspnet_Users.LastActivityDate` value falls within a specified time window. This time window value defaults to 15 minutes but can be customized via the `<membership>` configuration element in `Web.config`.

What is UTC Time?

Coordinated Universal Time, or UTC time, the standard international time that all time zones are expressed as offsets of. UTC does not get adjusted for daylight savings. To compute local time from UTC, simply add the time zone offset and then add an additional hour if daylight savings time is in effect. UTC time is commonly used to store date/time values in database systems because it is not tied to the database server's time zone.

For more information on the advantages of storing dates and times using UTC, as well as for information on how to work with UTC in SQL Server and .NET applications, refer to Using Coordinated Universal Time (UTC) to Store Date/Time Values.

Returning the Number of Users Currently Online

Let's create a simple example that shows the `GetNumberOfUsersOnline` method in action. At the end of this article you can download a complete working demo application that supports user accounts through the `SqlMembershipProvider`; the necessary setup and configuration for using the Membership system was discussed in previous installments of this article series. In the demo's `Default.aspx` page you'll find a Label Web control named `NumOnline`:

```
There are currently <asp:Label ID="NumOnline" runat="server" /> users logged on right now!
```

This Label's `Text` property is assigned the numeric value returned by the `GetNumberOfUsersOnline` method:

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
   If Not Page.IsPostBack Then
      'Display the number of users currently online
      NumOnline.Text = Membership.GetNumberOfUsersOnline()
   End If
End Sub
```

Keep in mind that this method only returns the number of *authenticated users*, which are users who have logged into the site. If there are 100 anonymous users visiting your site, and 20 logged in users, the `GetNumberOfUsersOnline` method will return 20.

As noted earlier, the `GetNumberOfUsersOnline` method determines whether a user is "logged in" or not based on the delta between the `aspnet_Users.LastActivityDate` value and the current date and time. By default, a 15 minute window is used, but this value can be customized. To change this window to 10 minutes, for example, go to your `Web.config` file and add a `userIsOnlineTimeWindow` attribute to the `<membership>` configuration element like so:

```
<?xml version="1.0"?>
<configuration>
   <system.web>
      ...

      <membership defaultProvider="CustomizedProvider" userIsOnlineTimeWindow="10">
```

```
        ...
      </membership>
   </system.web>
</configuration>
```

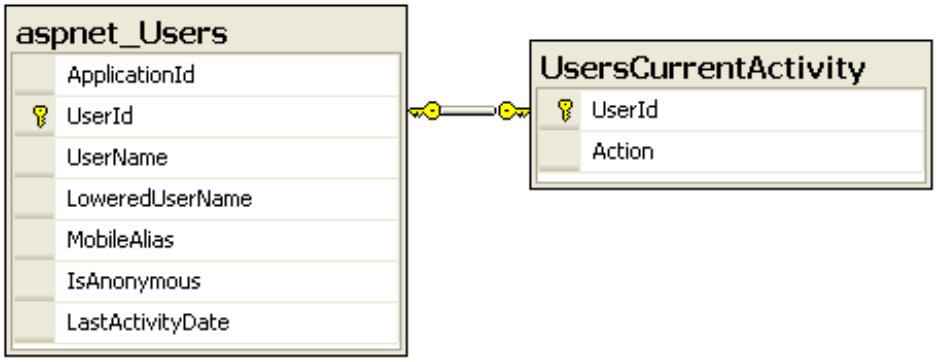
Tracking Additional Information About a User's Last Activity

The Membership system's built-in date/time tracking allows us to determine how many authenticated users are currently logged on, but it does not provide any further information. With a little bit of work we can extend this functionality to include a short description as to each user's most recent activity. For example, rather than just displaying the number of authenticated users who are logged onto the site, we could display a grid listing each user and what page on the site she is currently viewing.

To accomplish this we need to create our own database table that associated a description for the user's last activity with their user account record in the `aspnet_Users` table. Because I am only interested in the most recent activity (and am not interested in maintaining a log of each authenticated user's activity), I created a table named `UsersCurrentActivity` that establishes a one-to-one correspondence with the `aspnet_Users` table. Specifically, I defined the `UsersCurrentActivity` table as having two rows:

- **`UserId`** - a column of type `uniqueidentifier` (so as to match the type for the `UserId` column in `aspnet_Users`); this column serves as the table's primary key.
- **`Action`** - a column of type `nvarchar(255)` that stores a brief description of the user's last action.

I then established a foreign key constraint from the `UsersCurrentActivity.UserId` column to the `aspnet_Users.UserId` column.



Logging the User's Activity

After creating the `UsersCurrentActivity` table, my next task was to create a stored procedure that, when called, would update a specified user's `aspnet_Users.LastActivityDate` column *and* update the corresponding row in `UsersCurrentActivity` with a specified `Action` value. I created a stored procedure named `sproc_UpdateUsersCurrentActivity` that performed these two tasks:

```
ALTER PROCEDURE dbo.sproc_UpdateUsersCurrentActivity
(
   @UserId      uniqueidentifier,
   @Action      nvarchar(255),
   @CurrentTimeUtc   datetime
)
AS

BEGIN TRY
    BEGIN TRANSACTION   -- Start the transaction

    -- Update the LastActivityDate in aspnet_Users
   UPDATE   dbo.aspnet_Users
   SET    LastActivityDate = @CurrentTimeUtc
   WHERE    @UserId = UserId
```

```
    -- Update (or insert) activity record for user
    IF EXISTS(SELECT 1 FROM UsersCurrentActivity WHERE UserId = @UserId)
       -- Row exists, so UPDATE
       UPDATE UsersCurrentActivity SET
          Action = @Action
       WHERE UserId = @UserId
    ELSE
       -- No such row exists, so INSERT
       INSERT INTO UsersCurrentActivity(UserId, Action)
       VALUES(@UserId, @Action)

    -- If we reach here, success!
    COMMIT
END TRY
BEGIN CATCH
    -- Whoops, there was an error
    IF @@TRANCOUNT > 0
     ROLLBACK

    -- Raise an error with the details of the exception
    DECLARE @ErrMsg nvarchar(4000), @ErrSeverity int
    SELECT @ErrMsg = ERROR_MESSAGE(),
        @ErrSeverity = ERROR_SEVERITY()

    RAISERROR(@ErrMsg, @ErrSeverity, 1)
END CATCH
```

The above stored procedure starts by updating the specified user's `aspnet_Users` table's `LastActivityDate` column value. Next, a check is made to see if there exists a record for the specified user in the `UsersCurrentActivity` table. If there already exists a record, an `UPDATE` statement is used to update the user's `Action` column value; If not, an `INSERT` statement adds a new record to the table.

Because these stored procedure includes multiple data modification statements, it is wise to place the data modification logic within the scope of a transaction to ensure atomicity. The rational behind this approach, as well as the T-SQL syntax for starting, commiting, and rolling back transactions, are a bit beyond the scope of this article. For more information see Maintaining Database Consistency with Transactions and Managing Transactions in SQL Server Stored Procedures.

Calling the `sproc_UpdateUsersCurrentActivity` Stored Procedure from an ASP.NET Page

Each time an authenticated user visits a page in the site, we want to execute the `sproc_UpdateUsersCurrentActivity` stored procedure, passing in an appropriate value for the `UsersCurrentActivity.Action` column. Because this likely needs to be called from every page, it makes sense to utilize a *base page class*. A base page class is a class that derives from the `Page` class in the `System.Web.UI` namespace and adds additional page-level functionality. Once a base page class has been created, we can update our ASP.NET pages' code-behind classes to derive from this custom base page class (rather than from the `Page` class in the `System.Web.UI` namespace) and then they'll all have access to this added functionality. See Using a Custom Base Page Class for Your ASP.NET Pages' Code-Behind Classes for more information on this technique.

Included in the download at the end of this article is a class named `BasePage` that defines a method named `LogActivity`. The `LogActivity` accepts a String input and, if the visitor is authenticated, the `sproc_UpdateUsersCurrentActivity` stored procedure is called passing in the currently logged on user's `UserId` value, the passed-in String parameter, and the current UTC date and time.

```
Imports System.Data
Imports System.Data.SqlClient

Public Class BasePage
```

```
Public Class BasePage
   Inherits System.Web.UI.Page

   Protected Sub LogActivity(ByVal action As String)
      'Only proceed if the user is authenticated
      If Request.IsAuthenticated Then
         'Get information about the currently logged on user
         Dim usr As MembershipUser = Membership.GetUser
         If usr Is Nothing Then
            'Whoops, we don't know who this user is!
            Exit Sub
         End If

         'Read in the user's UserId value
         Dim UserId As Guid = CType(usr.ProviderUserKey, Guid)


         'Call the sproc_UpdateUsersCurrentActivity sproc
         Using myConnection As New
SqlConnection(ConfigurationManager.ConnectionStrings("MembershipConnectionString").ConnectionString)
            Dim myCommand As New SqlCommand("sproc_UpdateUsersCurrentActivity", myConnection)
            myCommand.CommandType = CommandType.StoredProcedure

            myCommand.Parameters.AddWithValue("@UserId", UserId)
            myCommand.Parameters.AddWithValue("@Action", action)
            myCommand.Parameters.AddWithValue("@CurrentTimeUtc", DateTime.UtcNow)

            'Execute the sproc
            myConnection.Open()
            myCommand.ExecuteNonQuery()
            myConnection.Close()
         End Using
      End If
   End Sub
End Class
```

Once this class has been created and the ASP.NET pages' code-behind classes derive from it, simply call `LogActivity` to record information about the user's current activity. For example, if you want to have the action "Visiting the site's homepage" recorded whenever a user visits the home page, have the home page's code-behind class derive from `BasePage` and then, in the `Page_Load` event handler, call `LogActivity("Visiting the site's homepage.")`.

The following code is from the code-behind class for `Default.aspx` (the home page).

```
Partial Class _Default
   Inherits BasePage

   Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
Me.Load
      MyBase.LogActivity("Visiting the site's homepage.")

      ...
   End Sub
End Class
```

Displaying Logged On Users and Their Current Actions

Now that we are recording each logged on user's last action, we can create a richer interface displaying information on the currently logged on users. I created a stored procedure named `sproc_GetUsersCurrentActivity` that returns the `UserName`, `Action`, and the number of minutes that have transpired since the users last activity for those users whose last activity time is within the specified window.

```
ALTER PROCEDURE dbo.sproc GetUsersCurrentActivity
```

```
(
   @ApplicationName         nvarchar(256),
   @MinutesSinceLastInActive int,
   @CurrentTimeUtc          datetime
)
AS

DECLARE @DateActive datetime
SELECT @DateActive = DATEADD(minute, -(@MinutesSinceLastInActive), @CurrentTimeUtc)

SELECT u.UserName,
      act.Action,
      DATEDIFF(minute, u.LastActivityDate, @CurrentTimeUtc) AS MinutesSinceAction

FROM   dbo.aspnet_Users u(NOLOCK),
      dbo.aspnet_Applications a(NOLOCK),
      dbo.UsersCurrentActivity act(NOLOCK)

WHERE u.ApplicationId = a.ApplicationId             AND
      LastActivityDate > @DateActive                AND
      a.LoweredApplicationName = LOWER(@ApplicationName) AND
      act.UserId = u.UserId

ORDER BY MinutesSinceAction ASC
```

As you can see, this stored procedure accepts three input parameters: `@ApplicationName`, `@MinutesSinceLastInActive`, and `@CurrentTimeUtc`. Because a single Membership user store can contain user information for multiple applications, we want to ensure that this stored procedure only returns information about logged in users for the specified application; this is the purpose of the `@ApplicationName` input parameter. The `@MinutesSinceLastInActive` and `@CurrentTimeUtc` parameters indicate the number of minutes a user is considered active since their last activity time in `aspnet_Users` and the current UTC date and time, respectively. These two parameters are used to compute the `@DateActive` time, which is the date and time threshold that separates currently logged on users from logged off users.

The results of this stored procedure can be displayed in an ASP.NET web page through a GridView. The `WhoIsOnline.aspx` page, which is part of the demo downloadable from the end of this article, provides an example. As the following screenshot illustrates, there are currently two users logged onto the site: Scott and Jisun. Scott is viewing the Who Is Online page while Jisun viewed the Users List page two minutes ago.

Shortcomings...

You are welcome to use this code and ideas in your websites, but be aware that there are a couple of limitations to keep in mind when evaluating using the Membership system's `GetNumberOfUsersOnline` method

and my enhancements.

The Membership system's `GetNumberOfUsersOnline` method and, by extension, my enhanced version only track *authenticated* users. If your site has pages that are accessible by anonymous users as well as authenticated users, however, you may want to also track the number of unauthenticated users (and where on the site they are currently visiting). Another shortcoming is that logging off does not affect the user's `LastActivityDate` column value. Consequently, if a user logs onto the site then immediately logs off, they'll count as being one of the site's logged on users for the next 15 minutes (or for however long you've set this time window).

These limitations have to do with the Membership system's behavior. In other words, these limitations were not just now added by our enhancements.

Conclusion

This article examined the Membership system's capabilities for indicating how many authenticated users are currently logged in. Behind the scenes, the Membership system tracks users' last activity date and includes a `GetNumberOfUsersOnline` method that returns the number of user accounts whose last activity date is within a certain interval. Unfortunately, the Membership system does not include any methods for returning a list of users that are logged in; nor does it provide any information as to what the currently logged in users are doing. With a little bit of work, we can add additional functionality that tracks the currently logged on user's current activity, such as what page they are visiting in the site.

Happy Programming!

- By Scott Mitchell

Further Reading

- Forms Authentication, Authorization, Membership, and Roles Tutorials (includes VB & C# versions!)
- Using Coordinated Universal Time (UTC) to Store Date/Time Values
- Maintaining Database Consistency with Transactions
- Managing Transactions in SQL Server Stored Procedures
- `TRY...CATCH` in SQL Server 2005
- Using a Custom Base Page Class for Your ASP.NET Pages' Code-Behind Classes
- Accessing and Updating Data in ASP.NET 2.0: Examining the Data Source Control's Events
- Tracking User Activity

Attachments

- Download the code used in this article

Article Information	
Article Title:	ASP.NET.Examining ASP.NET's Membership, Roles, and Profile - Part 10
Article Author:	Scott Mitchell
Published Date:	April 16, 2008
Article URL:	http://www.4GuysFromRolla.com/articles/041608-1.aspx